Make Linux better and faster with PAPI and Perf
All about Linux PAPI and Perf

Jan Altenberg
Open Source Automation Development Lab (OSADL) eG

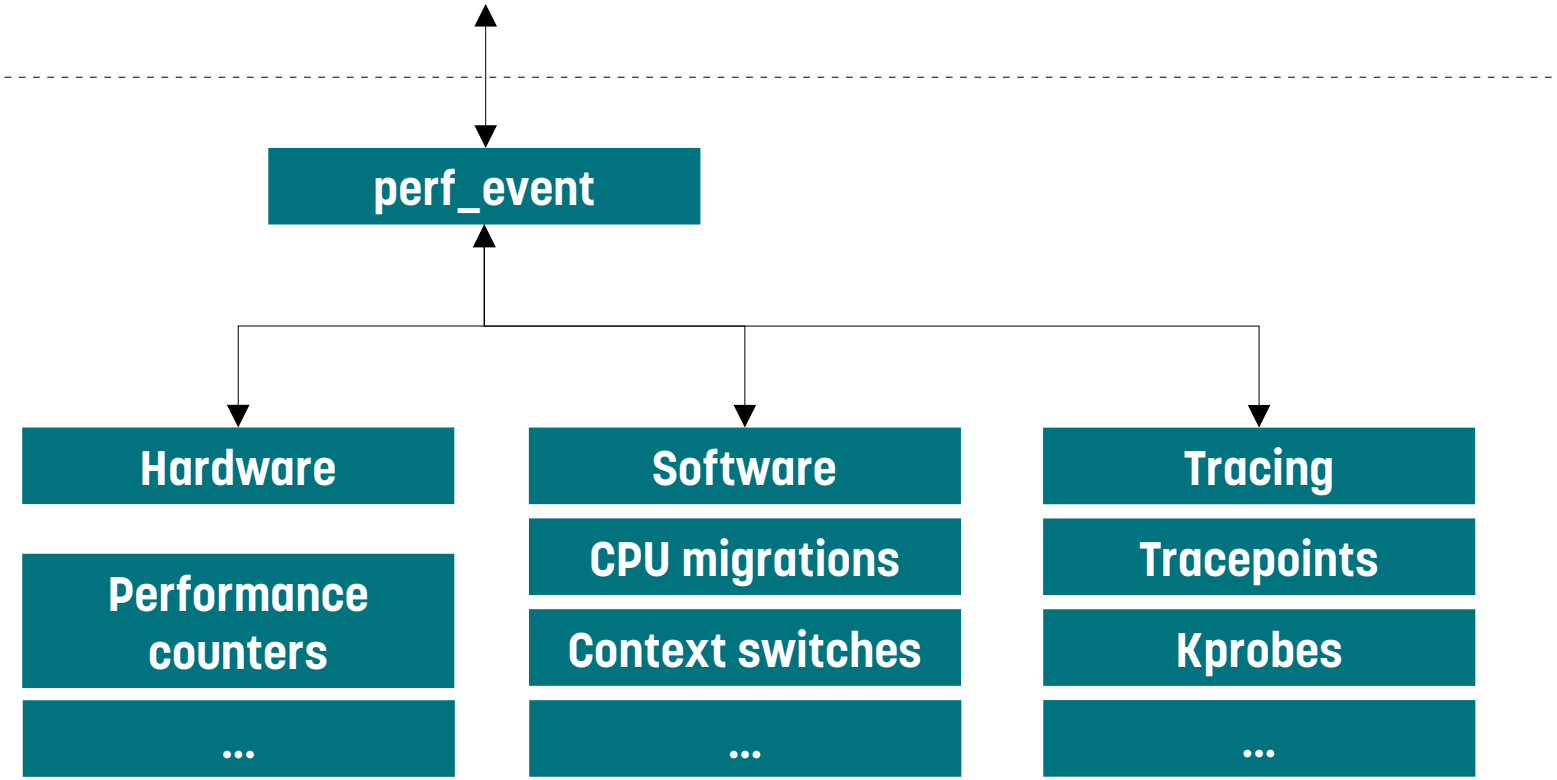
Hardware performance counters

- Modern CPUs come with a so called Performance Monitoring Unit (PMU).
- The PMU provides a special set of registers / counters that can directly count hardware events, such as cycles, instructions and branch misses.

Perf: Overview

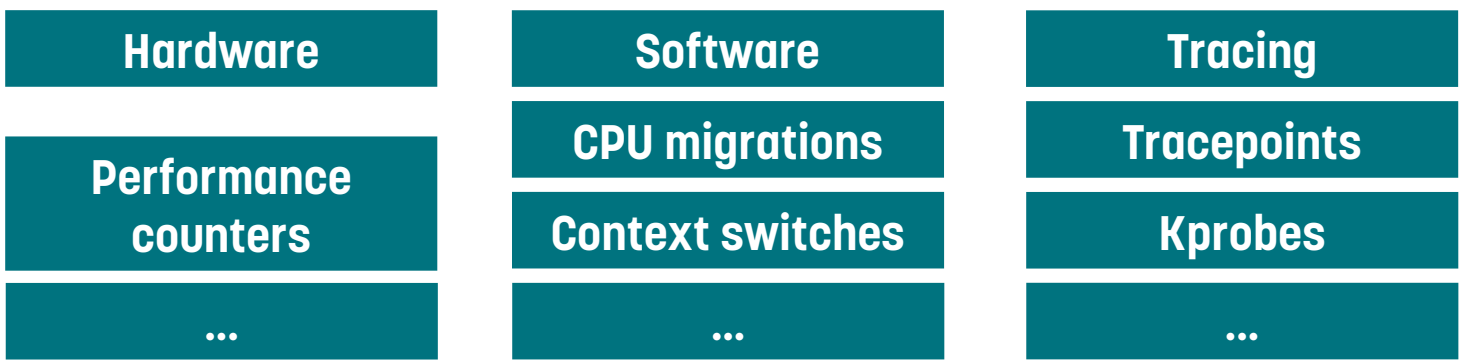
- Originally developed for using the subsystem for performance counters in Linux
- Over the years it got extended to profile and trace everything from the application over the kernel down to the hardware.

The Linux performance counters subsystem



The Linux performance counters subsystem

```
$ perf list
  branch-instructions OR branches                    [Hardware event]
  branch-misses                                      [Hardware event]
  [...]
  context-switches OR cs                             [Software event]
  cpu-clock                                          [Software event]
  [...]
  sched:sched_switch                                 [Tracepoint event]
  sched:sched_wait_task                              [Tracepoint event]
```



Perf: How to build it

```
$ cd tools/perf
$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux- prefix=/path_to_target_rfs/usr/ install
Auto-detecting system features:
...                                  dwarf: [ OFF ]
...                        dwarf_getlocations: [ OFF ]
...                                    glibc: [ on  ]
...                                   libbfd: [ OFF ]
...                              libbfd-buildid: [ OFF ]
[...]
...                                    libcap: [ OFF ]
...                                    libelf: [ OFF ]
[...]
...                                  get_cpuid: [ OFF ]
...                                        bpf: [ on  ]
...                                    libaio: [ on  ]
...                                  libzstd: [ OFF ]
```

Perf: How to build it

```
$ cd tools/perf
$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux- prefix=/path_to_target_rfs/usr/ install
Auto-detecting system features:
...                                  dwarf: [ OFF ]
...                        dwarf_getlocations: [ OFF ]
...                                    glibc: [ on  ]
...                                   libbfd: [ OFF ]
...                              libbfd-buildid: [ OFF ]
[...]
...                                    libcap: [ OFF ]
...                                    libelf: [ OFF ]
[...]
...                                  get_cpuid: [ OFF ]
...                                        bpf: [ on  ]
...                                     libaio: [ on  ]
...                                   libzstd: [ OFF ]
```

Perf: How to use it

- Access to the related operations might be limited to privileged processes.
- For non privileged processes (without CAP_PERFMON, CAP_SYS_PTRACE or CAP_SYS_ADMIN) access can be granted / limited via:

```
/proc/sys/kernel/perf_event_paranoid
```

Perf: How to use it

```
$ cat /proc/sys/kernel/perf_event_paranoid
2
```

Perf: How to use it

```
$ cat /proc/sys/kernel/perf_event_paranoid
2
```

-1	No access restrictions are imposed
>=0	Raw tracepoints and ftrace function tracepoints are excluded
>=1	System wide performance monitoring is excluded, capturing of user AND kernel events is possible
>=2	System wide performance monitoring is excluded, capturing of user events is possible

Perf: How to use it

```
$ cat /proc/sys/kernel/perf_event_paranoid
2
```

-1	No access restrictions are imposed
>=0	Raw tracepoints and ftrace function tracepoints are excluded
>=1	System wide performance monitoring is excluded, capturing of **user AND kernel** events is possible
>=2	System wide performance monitoring is excluded, capturing of **user** events is possible

Perf: How to use it

```
$ perf
 usage: perf [--version] [--help] [OPTIONS] COMMAND [ARGS]

 The most commonly used perf commands are:
   annotate        Read perf.data (created by perf record) and display annotated code
   [...]
   data            Data file related processing
   diff            Read perf.data files and display the differential profile
   [...]
   record          Run a command and record its profile into perf.data
   report          Read perf.data (created by perf record) and display the profile
   sched           Tool to trace/measure scheduler properties (latencies)
   script          Read perf.data (created by perf record) and display trace output
   stat            Run a command and gather performance counter statistics
   [...]
   timechart       Tool to visualize total system behavior during a workload
   top             System profiling tool
   version         display the version of perf binary
   probe           Define new dynamic tracepoints
   trace           strace inspired tool
```

Perf: How to use it

```
$ perf
 usage: perf [--version] [--help] [OPTIONS] COMMAND [ARGS]

 The most commonly used perf commands are:
   annotate        Read perf.data (created by perf record) and display annotated code
   [...]
   data            Data file related processing
   diff            Read perf.data files and display the differential profile
   [...]
   record          Run a command and record its profile into perf.data
   report          Read perf.data (created by perf record) and display the profile
   sched           Tool to trace/measure scheduler properties (latencies)
   script          Read perf.data (created by perf record) and display trace output
   stat            Run a command and gather performance counter statistics
   [...]
   timechart       Tool to visualize total system behavior during a workload
   top             System profiling tool
   version         display the version of perf binary
   probe           Define new dynamic tracepoints
   trace           strace inspired tool
```

perf stat

```
$ perf stat dd if=/dev/zero of=/dev/null bs=1M count=32
              9.72 msec task-clock                       #    0.599 CPUs utilized
                 4      context-switches                 #  411.343 /sec
                 0      cpu-migrations                   #    0.000 /sec
               320      page-faults                      #   32.907 K/sec
        15,508,154      cycles                           #    1.595 GHz
        20,506,426      instructions                     #    1.32  insn per cycle
         4,660,074      branches                         #  479.222 M/sec
            43,136      branch-misses                    #    0.93% of all branches
```

perf stat

```
$ perf stat dd if=/dev/zero of=/dev/null bs=1M count=...
              9.72 msec task-clock                #    ...CPUs utilized
                 4      context-switches          #    ...  /sec
                 0      cpu-migrations            #    0.000 /sec
               320      page-faults               #   32.907 K/sec
        15,508,154      cycles                    #    1.595 GHz
        20,506,426      instructions              #    1.32  insn per cycle
         4,660,074      branches                  #  479.222 M/sec
            43,136      branch-misses             #    0.93% of all branches
```

perf stat

```
$ perf stat dd if=/dev/zero of=/dev/null bs=1M count=32
              9.72 msec task-clock                       #    0.599 CPUs utilized
                 4      context-switches                 #  411.343 /sec
                 0      cpu-migrations                   #    0.000 /sec
               320      page-faults                      #   32.907 K/sec
        15,508,154      cycles                           #    1.595 GHz
        20,506,426      instructions                     #    1.32  insn per cycle
         4,660,074      branches                         #  479.222 M/sec
            43,136      branch-misses                    #    0.93% of all branches
```

perf stat

```
$ perf stat dd if=/dev/zero of=/dev/null bs=1M count=32
              9.72 msec task-clock                #    0.599 CPUs utilized
                 4      context-switches          #  411.343 /sec
                 0      cpu-migrations            #    0.000 /sec
               320      page-faults               #   32.907 K/sec
        15,508,154      cycles                    #    1.595 GHz
        20,506,426      instructions              #    1.32  insn per cycle
         4,660,074      branches                  #  479.222 M/sec
            43,136      branch-misses             #    0.93% of all branches
```

perf stat

```
$ perf stat -e cycles dd if=/dev/zero of=/dev/null bs=1M count=32

       14,737,704      cycles
```

perf stat

```
$ perf stat -e cycles,cycles:k,cycles:u dd if=/dev/zero of=/dev/null bs=1M count=32

       14,615,534      cycles
       13,820,592      cycles:k
          794,942      cycles:u
```

perf stat

```
$ perf stat -e cycles,cycles:k,cycles:u dd if=/dev/zero of=/dev/null bs=1M count=32

      14,615,534      cycles
      13,820,592      cycles:k
         794,942      cycles:u
```

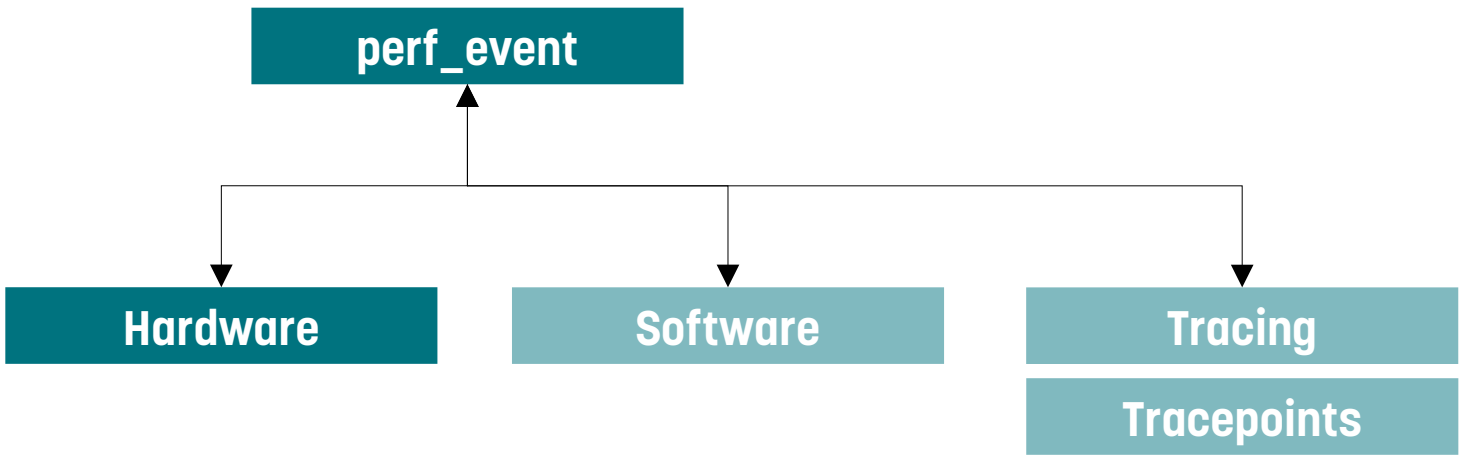
u	User
k	Kernel
h	Hypervisor
H	Host in a virtualized environment
G	Guest in a virtualized environment

perf stat

```
$ perf stat -e cycles,context-switches,sched:sched_switch sleep 2

         2,839,483      cycles
                 3      context-switches
                 3      sched:sched_switch
```

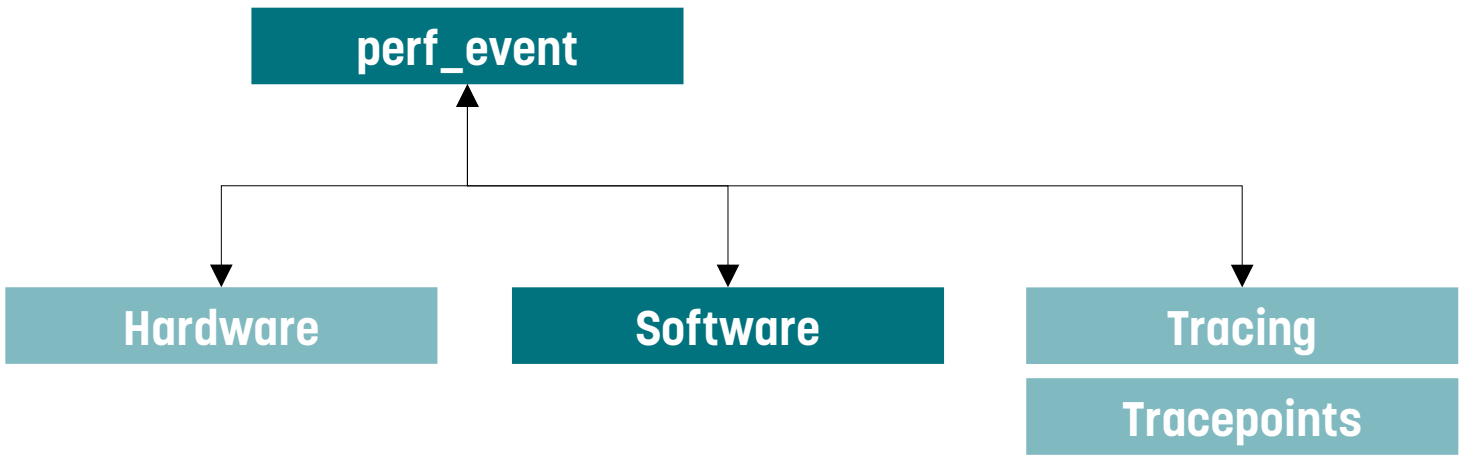
perf stat

```
$ perf stat -e cycles,context-switches,sched:sched_switch sleep 2

         2,839,483      cycles
                 3      context-switches
                 3      sched:sched_switch
```



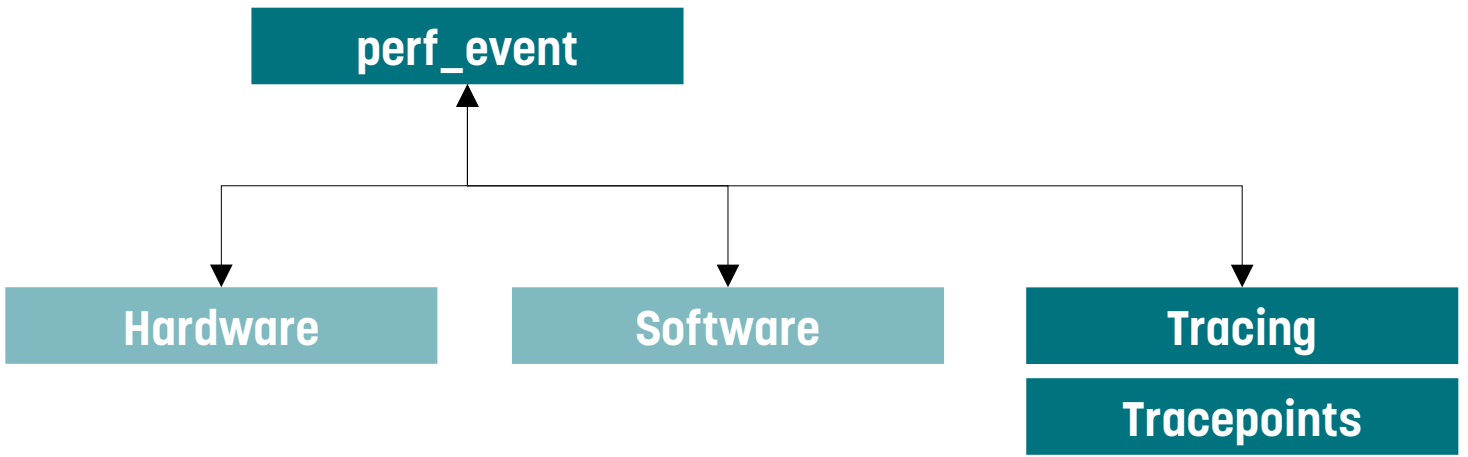
perf stat

```
$ perf stat -e cycles,context-switches,sched:sched_switch sleep 2

         2,839,483      cycles
                 3      context-switches
                 3      sched:sched_switch
```



perf stat

```
$ perf stat -e cycles,context-switches,sched:sched_switch sleep 2

         2,839,483      cycles
                 3      context-switches
                 3      sched:sched_switch
```



perf probe

```
$ perf probe ktime_get
Added new event:
  probe:ktime_get      (on ktime_get)

You can now use it in all perf tools, such as:

     perf record -e probe:ktime_get -aR sleep 1
```

perf probe

```
$ perf probe ktime_get
Added new event:
  probe:ktime_get      (on ktime_get)

You can now use it in all perf tools, such as:

     perf record -e probe:ktime_get -aR sleep 1

$ perf stat -e probe:ktime_get sleep 1
                 1      probe:ktime_get
```

perf probe

```
$ perf probe ktime_get
Added new event:
  probe:ktime_get      (on ktime_get)

You can now use it in all perf tools, such as:

     perf record -e probe:ktime_get -aR sleep 1

$ perf stat -e probe:ktime_get sleep 1
                 1      probe:ktime_get

$ perf stat -a -e probe:ktime_get sleep 1
               433      probe:ktime_get
```

perf probe

```
$ perf probe ktime_get
Added new event:
  probe:ktime_get      (on ktime_get)

You can now use it in all perf tools, such as:

     perf record                                  sleep 1

$ perf stat -e probe                              p 1
                1      probe:ktime_get

$ perf stat -a -e probe:ktime_get sleep 1
              433      probe:ktime_get
```

perf probe

```
$ perf probe -x /lib/aarch64-linux-gnu/libc.so.6 printf
Added new event:
  probe_libc:printf    (on printf in /usr/lib/aarch64-linux-gnu/libc.so.6)

You can now use it in all perf tools, such as:

perf record -e probe_libc:printf -aR sleep 1
```

perf probe

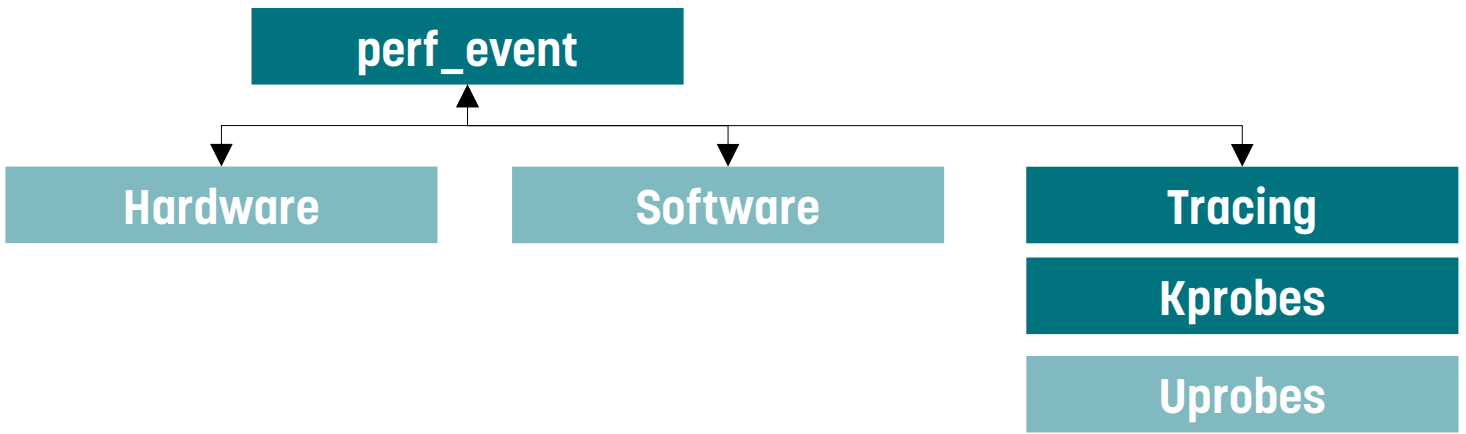
```
int main(void)
{
    int count = 0;

    for (count = 0; count < 10; count++)
        printf("Hello COOL %d\n", count);

    return 0;
}

$ gcc -Wall -o hello hello.c
```

perf probe

```
$ perf stat -e probe_libc:printf ./hello

Hello COOL 0
Hello COOL 1
Hello COOL 2
[...]
Hello COOL 7
Hello COOL 8
Hello COOL 9

 Performance counter stats for './hello':

                10      probe_libc:printf
```

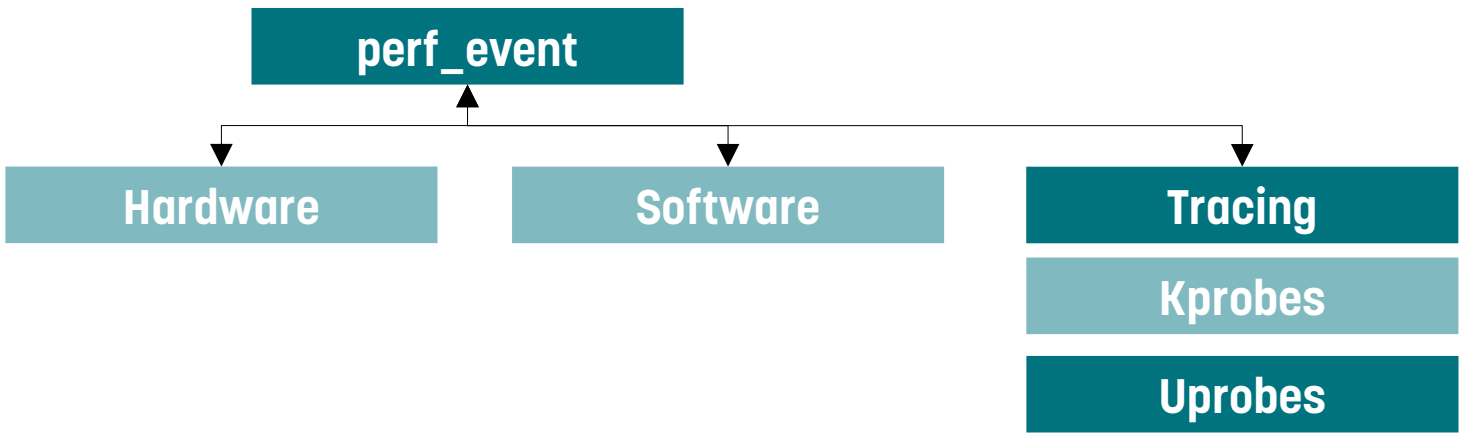
perf stat

```
$ perf stat -e cycles,context-switches,sched:sched_switch,\
  probe:ktime_get,probe_libc:printf ./hello

        2,601,559      cycles
                2      context-switches
                2      sched:sched_switch
                2      probe:ktime_get
               10      probe_libc:printf
```



perf stat

```
$ perf stat -e cycles,context-switches,sched:sched_switch,\
  probe:ktime_get,probe_libc:printf ./hello

        2,601,559      cycles
                2      context-switches
                2      sched:sched_switch
                2      probe:ktime_get
               10      probe_libc:printf
```



perf record

```
$ perf record -a -e cycles,context-switches,sched:sched_switch,\
  probe:ktime_get,probe_libc:printf ./hello

Hello COOL 0
Hello COOL 1
Hello COOL 2
[...]
Hello COOL 7
Hello COOL 8
Hello COOL 9

[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.076 MB perf.data (157 samples) ]
```

perf report

```
$ perf report

# Total Lost Samples: 0
# Samples: 44  of event 'cycles'
# Event count (approx.): 4818926
#
# Overhead  Command  Shared Object          Symbol
# ........  .......  .....................  ................................
#
    12.00%  perf-ex  [kernel.kallsyms]      [k] unmap_page_range
    11.10%  hello    [kernel.kallsyms]      [k] mas_wr_walk
    10.66%  hello    [kernel.kallsyms]      [k] memset
[...]
# Samples: 19  of event 'context-switches'
# Event count (approx.): 30
# Overhead  Command          Shared Object      Symbol
# ........  ...............  .................  ..................
#
    40.00%  swapper          [kernel.kallsyms]  [k] schedule_idle
    13.33%  perf             [kernel.kallsyms]  [k] preempt_schedule
    10.00%  hello            [kernel.kallsyms]  [k] do_task_dead
    10.00%  kworker/u8:3-ev  [kernel.kallsyms]  [k] schedule
    10.00%  rcu_preempt      [kernel.kallsyms]  [k] schedule
[...]
```

perf trace

```
$ perf trace ./hello

[...]
0.986 ( 0.043 ms): hello/569 munmap(addr: 0xffff8c1c5000, len: 20962)                    = 0
1.141 ( 0.007 ms): hello/569 getrandom(ubuf: 0xffff8c186970, len: 8, flags: NONBLOCK)    = 8
1.157 ( 0.005 ms): hello/569 brk()                                                       = 0xaaaad66c2000
1.167 ( 0.013 ms): hello/569 brk(brk: 0xaaaad66e3000)                                    = 0xaaaad66e3000
1.216 ( 0.032 ms): hello/569 write(fd: 1, buf: 0xaaaad66c22a0, count: 13)                 = 13
1.255 ( 0.009 ms): hello/569 write(fd: 1, buf: 0xaaaad66c22a0, count: 13)                 = 13
[...]
1.336 ( 0.010 ms): hello/569 write(fd: 1, buf: 0xaaaad66c22a0, count: 13)                 = 13
1.352 ( 0.010 ms): hello/569 write(fd: 1, buf: 0xaaaad66c22a0, count: 13)                 = 13
1.368 ( 0.010 ms): hello/569 write(fd: 1, buf: 0xaaaad66c22a0, count: 13)                 = 13
1.384 ( 0.010 ms): hello/569 write(fd: 1, buf: 0xaaaad66c22a0, count: 13)                 = 13
1.422 (         ): hello/569 exit_group()                                                = ?
```

perf top

```
PerfTop: 94 irqs/sec  kernel:88.3%  exact:  0.0% lost: 0/0 drop: 0/0 [4000Hz cycles],  (all, 4 CPUs)
----------------------------------------------------------------------------------------------------

     6.20%  perf           [.] rb_next
     5.35%  [kernel]       [k] __softirqentry_text_start
     4.32%  [kernel]       [k] fec_enet_rx_napi
     3.31%  [kernel]       [k] kallsyms_expand_symbol.constprop.0
     3.29%  perf           [.] __symbols__insert
     3.24%  perf           [.] kallsyms__parse
     2.71%  libc.so.6      [.] strchr
     2.60%  [kernel]       [k] format_decode
     2.60%  [kernel]       [k] number
     2.59%  [kernel]       [k] finish_task_switch.isra.0
     2.30%  [kernel]       [k] _raw_spin_unlock_irqrestore
     2.27%  [kernel]       [k] vsnprintf
     1.73%  [kernel]       [k] update_iter
     1.69%  libc.so.6      [.] _int_malloc
     1.60%  libc.so.6      [.] __aarch64_swp4_rel
     1.56%  libc.so.6      [.] __libc_calloc
```

perf top

```
PerfTop: 94 irqs/sec  kernel:88.3%  exact:  0.0% lost: 0/0 drop: 0/0 [4000Hz cycles],  (all, 4 CPUs)
-------------------------------------------------------------------------------------------------

     6.20%  perf           [.] rb_next
     5.35%  [kernel]       [k] __softirqentry_text_start
     4.32%  [kernel]       [k] fec_enet_rx_napi
     3.31%  [kernel]       [k] kallsyms_expand_symbol.constprop.0
     3.29%  perf           [.] __symbols__insert
     3.24%  perf           [.] kallsyms__parse
     2.71%  libc.so.6      [.] strchr
     2.60%  [kernel]       [k] format_decode
     2.60%  [kernel]       [k] number
     2.59%  [kernel]       [k] finish_task_switch.isra.0
     2.30%  [kernel]       [k] _raw_spin_unlock_irqrestore
     2.27%  [kernel]       [k] vsnprintf
     1.73%  [kernel]       [k] update_iter
     1.69%  libc.so.6      [.] _int_malloc
     1.60%  libc.so.6      [.] __aarch64_swp4_rel
     1.56%  libc.so.6      [.] __libc_calloc
```

LET'S DO A FLOOD PING TO THE DEVICE

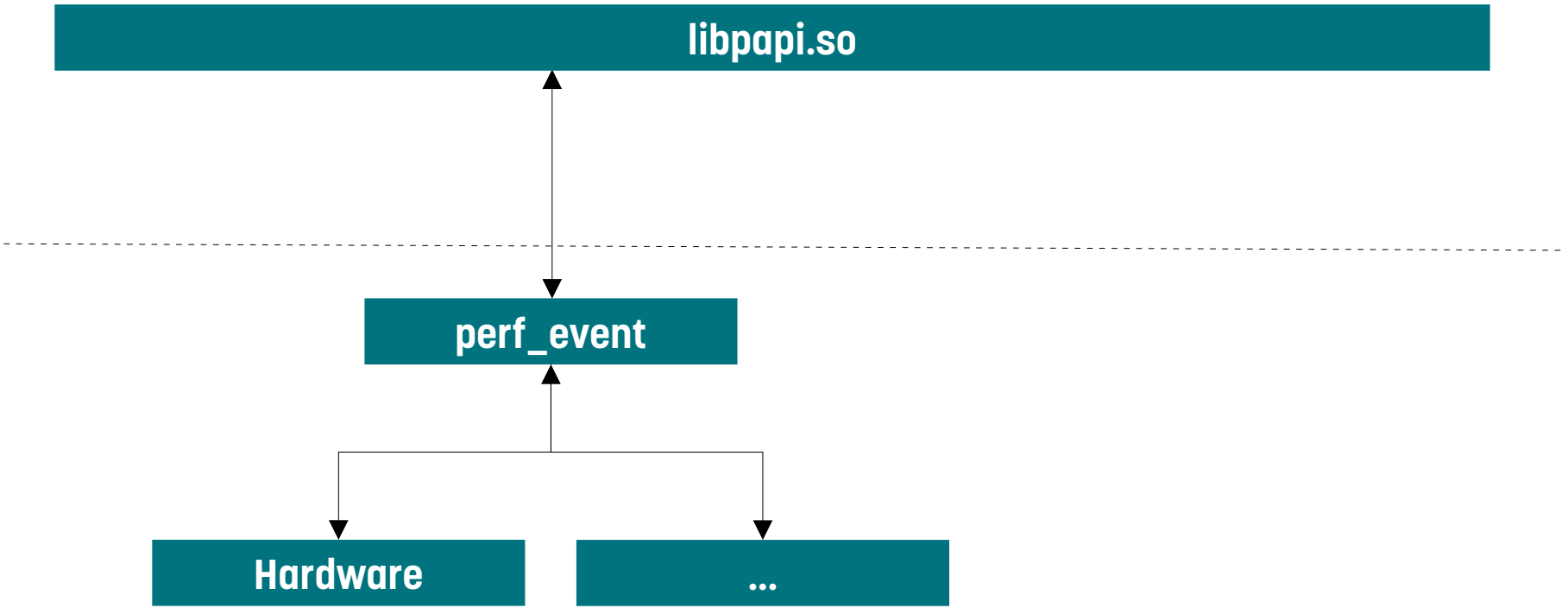
perf top

```
PerfTop: 1098 irqs/sec  kernel:96.6%  exact:  0.0% lost: 0/0 drop: 0/0 [4000Hz cycles],  (all, 4 CPUs)
-------------------------------------------------------------------------------------------------------

    14.09%  [kernel]       [k] fec_enet_rx_napi
    12.74%  [kernel]       [k] __softirqentry_text_start
     5.84%  [kernel]       [k] fec_enet_start_xmit
     5.14%  [kernel]       [k] memmove
     1.45%  [kernel]       [k] finish_task_switch.isra.0
     1.29%  perf           [.] sort__sym_cmp
     1.28%  perf           [.] dso__find_symbol
     1.16%  perf           [.] perf_hpp__is_dynamic_entry
     1.14%  [kernel]       [k] preempt_count_sub
     1.12%  perf           [.] hists__findnew_entry
     1.09%  [kernel]       [k] default_idle_call
     0.98%  [kernel]       [k] __ip_append_data
     0.89%  [kernel]       [k] check_preemption_disabled
     0.88%  [kernel]       [k] preempt_count_add
```

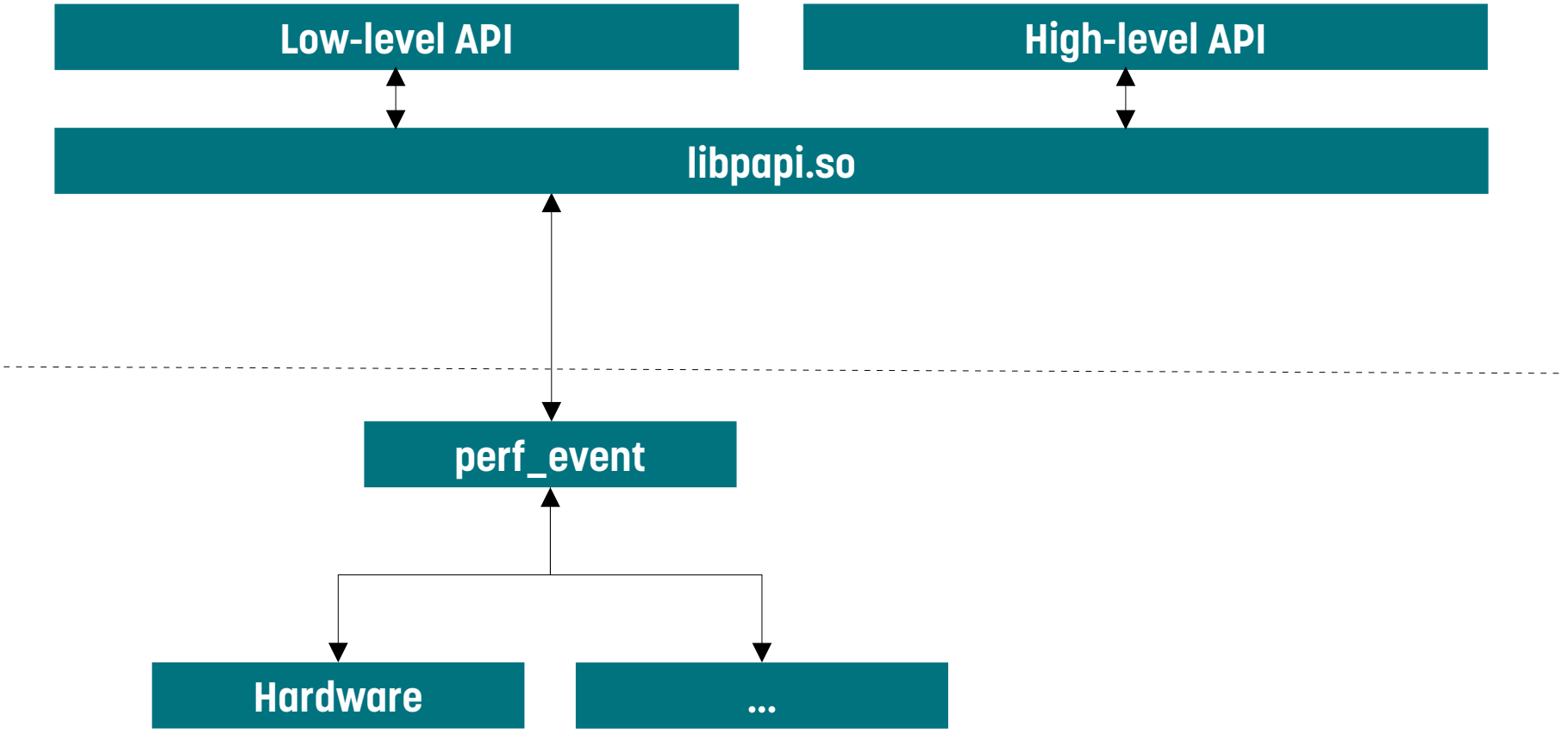
PAPI: Performance API

- Library that provides a consistent interface to hardware performance counters and other methodologies for performance measurements
- Available for multiple operating systems
- On Linux it provides a perf_event component
- Latest release available at: https://icl.utk.edu/papi/software/index.html

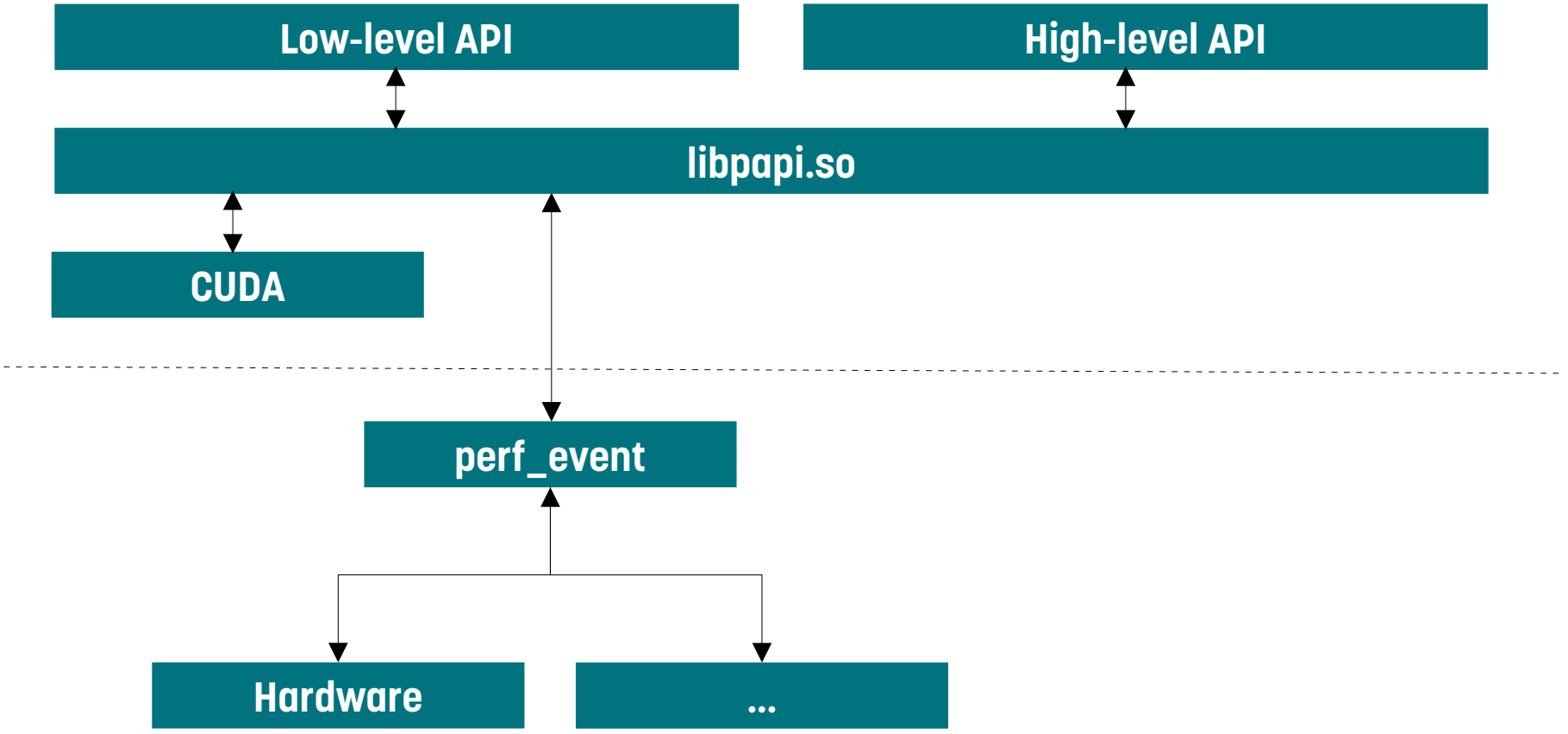
PAPI: Overview



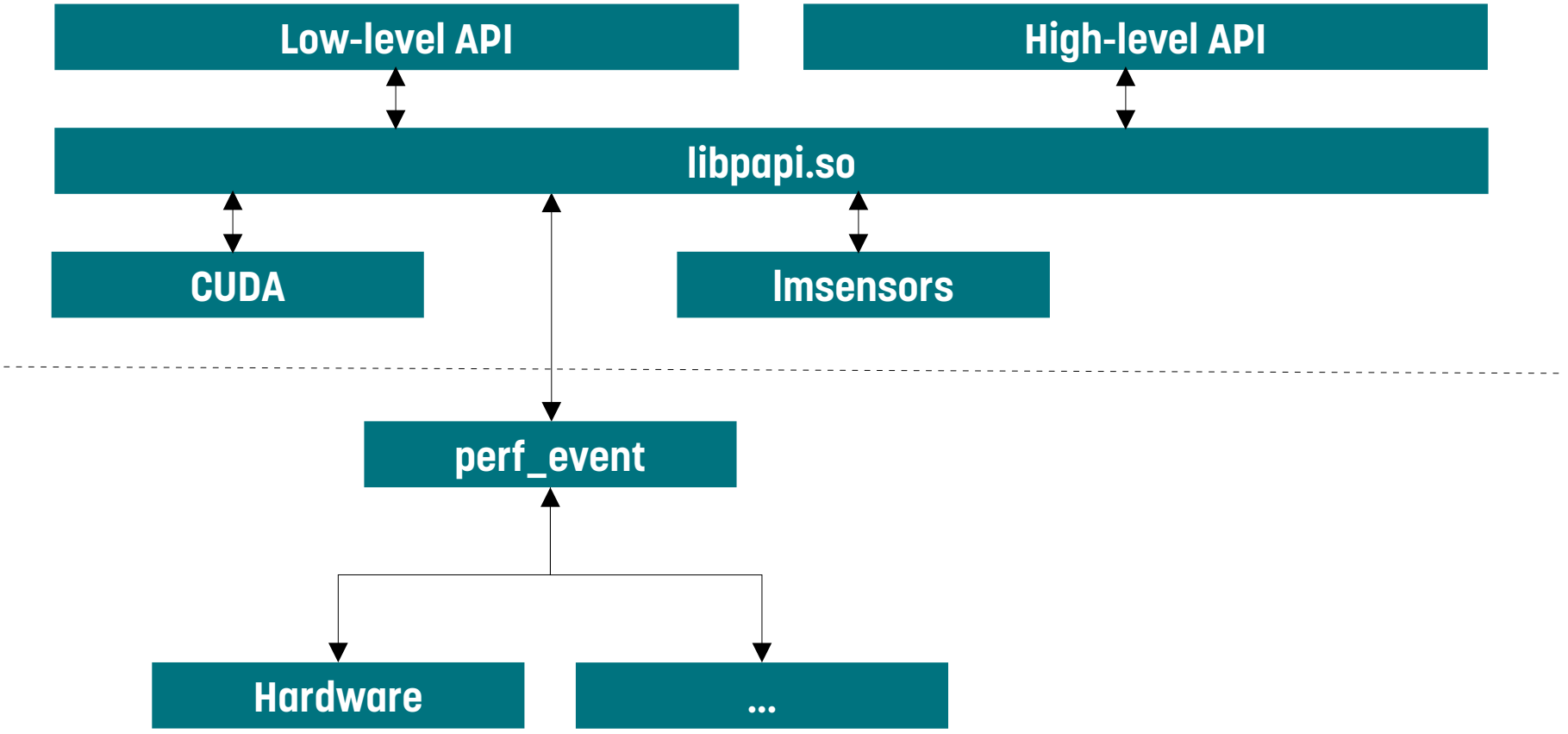
PAPI: Overview



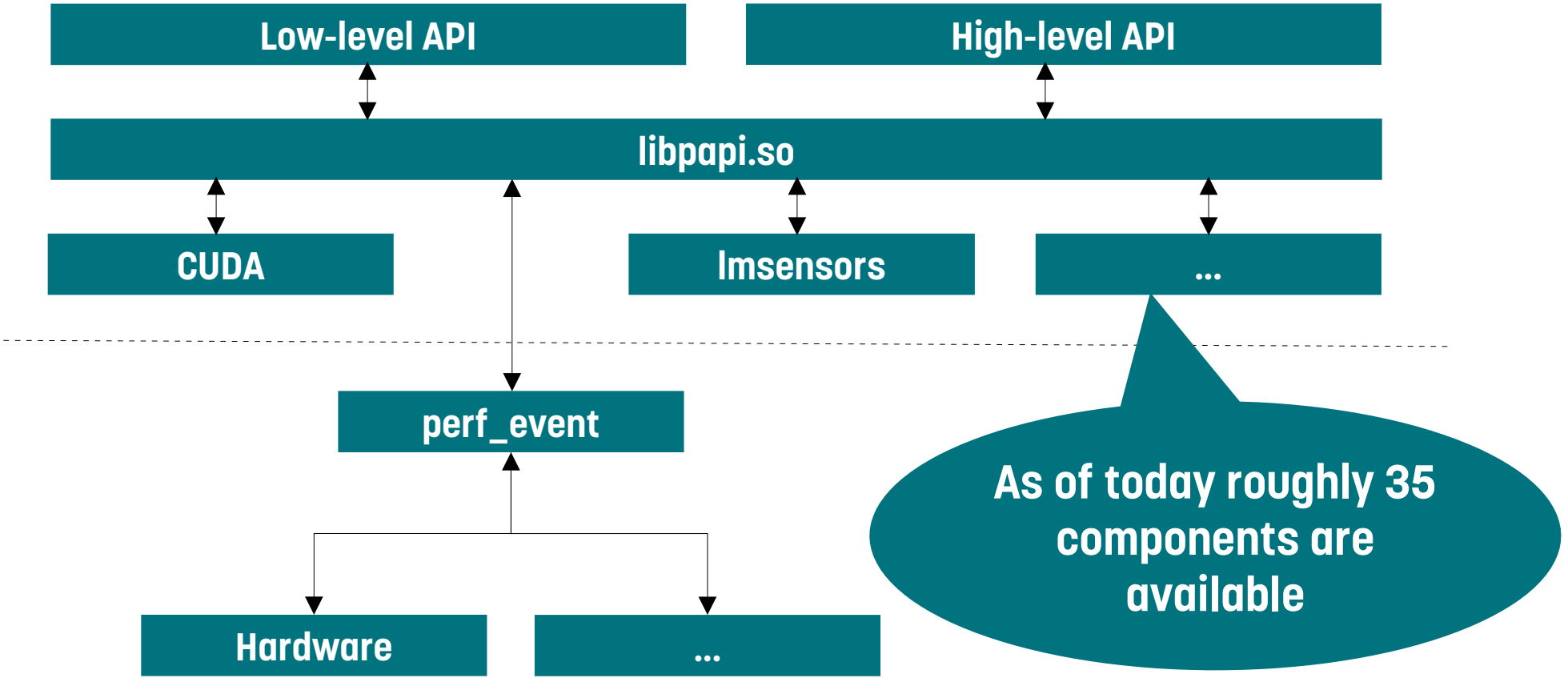
PAPI: Overview



PAPI: Overview



PAPI: Overview



PAPI: Event types

- Native events:
  Available events on a specific platform
- Preset events:
  Pre-defined set of commonly used events with a unique naming scheme (these are mapped to native events)
- Derived events:
  Events which are derived from multiple native events

PAPI: How to build it

```
$ ./configure \
  --with-CPU=arm \
  --with-arch=aarch64 \
  --with-ffsll \
  --host=aarch64-linux \
  --with-walltimer=clock_realtime \
  --with-tls=__thread \
  --with-virtualtimer=clock_thread_cputime_id
$ make
$ make DESTDIR=/path_to_target_rfs
```

PAPI: Show available components

```
$ papi_component_avail

Available native events and hardware information.
--------------------------------------------------------
PAPI version             : 7.0.0.0
Operating system         : Linux 6.1.4
Vendor string and code   : ARM_ARM (65, 0x41)
Model string and code    : ARM Cortex A53 (4, 0x4)
CPU revision             : 4.000000
[...]
```

PAPI: Show available components

```
Compiled-in components:
Name:   perf_event              Linux perf_event CPU counters
Name:   perf_event_uncore       Linux perf_event CPU uncore and
northbridge
   \-> Disabled: No uncore PMUs or events found
Name:   sysdetect               System info detection component

Active components:
Name:   perf_event              Linux perf_event CPU counters
                                Native: 220, Preset: 14, Counters: 6
                                PMUs supported: perf, perf_raw, arm_ac53

[...]
```

PAPI: Show available components

```
Compiled-in components:
Name:   perf_event                      Linux perf_event CPU counters
Name:   perf_event_uncor                PU uncore and
northbridge
   \-> Disabled: No u
Name:   sysdetect                       n component

Active components:
Name:   perf_event                      Linux perf_event CPU counters
                                        Native: 220, Preset: 14, Counters: 6
                                        PMUs supported: perf, perf_raw, arm_ac53

[..]
```

Adjustable via --with-components when configuring the build

PAPI: Show available preset events

```
================================================================================
  PAPI Preset Events
================================================================================
    Name         Code    Avail Deriv Description (Note)
PAPI_L1_DCM  0x80000000  Yes   Yes   Level 1 data cache misses
PAPI_L1_ICM  0x80000001  Yes   No    Level 1 instruction cache misses
[...]
PAPI_TOT_CYC 0x8000003b  Yes   No    Total cycles
PAPI_LST_INS 0x8000003c  No    No    Load/store instructions completed
[...]
```

PAPI: Low-level API

```
#define LOOPS 1000000

int ret;
int eventset = PAPI_NULL;
int test = LOOPS;
long long icount, lcount = 0;

/* Initialize the PAPI libary */
ret = PAPI_library_init(PAPI_VER_CURRENT);

if (ret != PAPI_VER_CURRENT) {
    printf(stderr, "Error initializing PAPI! %s\n", PAPI_strerror(ret));
    return 0;
}
```

PAPI: Low-level API

```
/* Create an eventset... */
ret = PAPI_create_eventset(&eventset);

if (ret != PAPI_OK) {
   fprintf(stderr, "Error creating eventset! %s\n", PAPI_strerror(ret));
}

/* ...and add an event */
ret = PAPI_add_named_event(eventset, "PAPI_TOT_CYC");

if (retval != PAPI_OK) {
   fprintf(stderr,"Error adding PAPI_TOT_CYC: %s\n",PAPI_strerror(ret));
}
```

PAPI: Low-level API

```
/* Reset and start the measurement */
PAPI_reset(eventset);
ret = PAPI_start(eventset);

if (ret != PAPI_OK) {
   fprintf(stderr, "Error starting loop: %s\n",
   PAPI_strerror(ret));
}

/* Do some work */
while (test--) lcount++;
```

PAPI: Low-level API

```
/* Stop the measurement */
ret = PAPI_stop(eventset, &icount);

/* Print the results */
printf("Executed a total of %lld loops\n", lcount);

if (retval != PAPI_OK)
   fprintf(stderr, "Error stopping: %s\n", PAPI_strerror(retval));
else
   printf("Measured %lld cycles\n", icount);
```

PAPI: Low-level API

```
$ gcc -O1 -Wall -o papi_lowlevel papi_lowlevel.c -lpapi
$ ./papi_lowlevel

Executed a total of 1000000 loops
Measured 1005981 cycles
```

PAPI: Low-level API

```
$ gcc -O1 -Wall -o papi_lowlevel papi_lowlevel.c -lpapi
$ ./papi_lowlevel

Executed a total of 1000000 loops
Measured 1005981 cycles

$ gcc -O2 -Wall -o papi_lowlevel papi_lowlevel.c -lpapi
$ ./papi_lowlevel

Executed a total of 1000000 loops
Measured 3049 cycles
```

PAPI: High-level API

- Simplified programming interface
- Eventsets can be defined with an environment variable
- Automated reporting

PAPI: High-level API

```
ret = PAPI_hl_region_begin("COOL example");
if (ret != PAPI_OK) {
   fprintf(stderr,"PAPI_hl_region_begin failed: %s\n",
              PAPI_strerror(ret));
}

while (test--)
   lcount++;

ret = PAPI_hl_region_end("COOL example");
if (ret != PAPI_OK) {
   fprintf(stderr,"PAPI_hl_region_end failed: %s\n",
              PAPI_strerror(ret));
}
```

PAPI: High-level API

```
$ gcc -O1 -Wall -o papi_highlevel papi_highlevel.c -lpapi
$ export PAPI_EVENTS="PAPI_TOT_CYC,PAPI_L1_ICM"
$ ./papi_highlevel

$ less papi_hl_output/rank_042349.json
[...]
          "name":"COOL example",
          "parent_region_id":"-1",
          "cycles":"5170",
          "real_time_nsec":"638000",
          "PAPI_TOT_CYC":"1020966",
          "PAPI_L1_ICM":"182"
[...]
```

PAPI: High-level API

```
$ gcc -O2 -Wall -o papi_highlevel papi_highlevel.c -lpapi
$ export PAPI_EVENTS="PAPI_TOT_CYC,PAPI_L1_ICM"
$ ./papi_highlevel

$ less papi_hl_output/rank_768929.json
[...]
          "name":"COOL example",
          "parent_region_id":"-1",
          "cycles":"136",
          "real_time_nsec":"10875",
          "PAPI_TOT_CYC":"18381",
          "PAPI_L1_ICM":"195"
[...]
```

Summary

- Perf offers a wide range of possibilities for profiling and tracing everything from the application over the kernel down to the hardware.
- PAPI offers an OS independent and consistent interface to several methodologies for performance measurements and system analysis.
- Both technologies can be used on all major architectures.
- Even though cross-compilation of these tools is possible, native building is highly recommended.